プログラムの自動安定化変換について

AUTHOR(S):

近藤, 祐史; 野田, 松太郎

CITATION:

近藤, 祐史 ...[et al]. プログラムの自動安定化変換について. 数理解析研究所講究録 2003, 1335: 181-188

ISSUE DATE:

2003-07

URL:

http://hdl.handle.net/2433/43355

数理解析研究所講究録 1335 巻 2003 年 181-187

プログラムの自動安定化変換について

近藤 祐史*
YUJI KONDOH
詫間電波工業高等専門学校†
TAKUMA NATIONAL COLLEGE OF TECHNOLOGY
野田 松太郎‡
MATU-TAROW NODA
愛媛大学工学部情報工学科
DEPERTMENT OF COMPUTER SCIENCE, EHIME UNIVERSITY

1 はじめに

白柳と Sweedler は、代数的アルゴリズムを近似計算で行ったときに起きる不安定な場合に対し、アルゴリズムの安定化手法を提案し理論の証明を行った [4]。この手法を用いることは、数値数式計算を行う一つの有効な方法である。安定化手法の基本は、

(STAB1) アルゴリズムの構造は変えない

(STAB2) データ領域において普通の係数を区間数に変える

(STAB3) 条件文等の述語の直前において 0 を含む区間数を 0 と見なすゼロ書換えを行う

である。これらにより変換されたアルゴリズムを、

(STAB4) 精度を上げながら再計算する

ことにより、真の解を得るかまたはそれに近づく解を得ることができる。最近の研究では、**(STAB3)** に関しては計算直後にゼロ書換えを行って安定化した例が報告されている [2]。しかし、この技術の適用範囲は未だ不十分である。そこで、既存のプログラムに対し、安定化手法が適用可能であるかどうかを検討することは非常に重要である。既存のプログラムを安定化手法を用いたものへ書換えることは、必ずしも容易ではない。そのため、自動的に安定化手法を用いたプログラムへ書換えるシステムの開発が必要である。そのようなシステムとして、関川らは数式処理システム Maple を用いて、Maple で書かれたプログラムの自動安定化を行うシステムを開発した [3]。また、筆者らは数式処理システム Risa/Asir によって、Asir 言語で書かれたプログラムに対して自動安定化を行うシステムを開発した [1]。これらは、数式処理のプログラムを自動安定化することを目的としている。

*kondoh@dc.takuma-ct.ac.jp
†平成 14 年度文部科学省内地研究員
‡noda@cs.ehime-u.ac.jp

本稿では、安定化手法の適用範囲の対象を従来の数式処理のアルゴリズムではなく数値計算とした場合の研究を遂行するために、数値計算プログラムを自動安定化するシステムの構築を目指す。本稿で述べるシステムは、実行環境を区間演算機能を有効にした Risa/Asir として、C 言語で書かれたプログラムを Asir 言語へ変換することにより実現する。そのため、C 言語を入力とし、安定化された Asir 言語プログラムを出力するトランスレータの開発を行った。

2 プログラムの自動安定化

安定化手法を施したプログラムの実行環境は、区間演算機能を有効にしている Risa/Asir[1] とする。区間演算機能を有効にしている Risa/Asir では、**(STAB3)** を実現するために、区間演算の直後にゼロ書換えを自動的に行う自動ゼロ書換え機能を実装している。そのため、この機能を使用するか否かによって対応する必要がある。

自動安定化には、安定化手法を用いたプログラムに書換えるトランスレータが必要である。このトランスレータの入力は C 言語とし、出力は安定化手法を用いた Asir 言語のプログラムとする。書換えは、アルゴリズムの安定化手法に従い、

- **(STAB1)** に関しては、プログラムの構造は変えない。そのため、コメント等も変えない
- **(STAB2)** に関しては、入力や浮動小数点数データを区間拡張する
- **(STAB3)** に関しては、計算直後にゼロ書換えを行うときは自動ゼロ書換え機能を有効にし、述語の直前に行うときには自動ゼロ書換え機能を無効にする。両方ともにそれぞれ対応した書換えをする
- **(STAB4)** に関しては、精度を上げながら再計算を行うため、再計算に役立つ補助プログラムを出力する。実行する際には、この補助プログラムだけでは十分ではない場合もあるが、これがあることによりユーザの負担を軽減できる

という方針で行う。具体的には、C 言語と Asir 言語の違いを吸収するため、また安定化のため次のような処理を行う。

- 識別子（変数名、関数名など）

  Asir では、変数名の先頭は大文字アルファベット、関数名の先頭は小文字アルファベットでなければならない制約がある。C 言語では、どちらでもいいため変数名と関数名を書換える必要がある。変数名には先頭に `V_` を付加し、関数名には先頭に `f_` を付加する。現在は固定であるが付加する文字列は、指定できるようにした方が良いと思われる。

- 定数

  整数（整定数） 8 進数への対応、L や U などの接尾子の削除

  文字、文字列 C 言語で行われている文字列の連結は行われないので、連結されたものに書換える。

  浮動小数点数 安定化手法のために精度を上げて再計算するときの精度指定のために、有理数化しかつ区間数化する。ここでも、接尾子は削除する。例えば、1.23 は、

  ```
  1.23  →  tointval(123*10^(-2))
  ```

  と書換える。ここで、`tointval` は区間数化するための関数である。

- 変数の宣言

  Asir 言語には、型の概念がないため変数の宣言は必要ない。そのため、基本的には削除すれば良い。しかし、初期化がある場合には代入文として書換える必要がある。また、配列については、Asir のベクトルへ書換える。1 次元の配列の場合は容易であるが、2 次元以上になると繁雑になるため、配列関係を処理する array 関数を作成し、array 関数を使用するように書換える。

- ポインタ変数の宣言

  Asir にはポインタがないため、プログラムとしてはポインタではない変数と同じにする。この書換えは、配列を関数へ引数として渡し、関数内では配列参照として記述されている場合を仮定している。そのため、ポインタ参照で値を参照したり代入したりしている場合は、動作が保証されない。これは、今後の課題である。

- 述語（関係演算子など）

  C 言語では、関係演算子 (<, <=, >, >=)、等値演算子 (==, ! =)、否定!が、データの比較のために使われている。これらは、安定化手法において、0 との比較に変更し、ゼロ書換えを行った上で 0 との比較を行う必要がある。そのため、元のプログラムが 0 との比較であった場合は、そのままであるが、0 との比較ではない場合には、移項し 0 との比較に書換える。また、区間演算を有効にした Asir では、自動ゼロ書換え機能が有効かどうかによって、プログラムの書換え処理を変える必要がある。

  例

  自動ゼロ書換え機能有効時

  ```
  a<b    →  V_a-(V_b)<0
  a<0    →  V_a<0
  a==0   →  V_a==0
  !a     →  ! V_a
  ```

  自動ゼロ書換え機能無効時

  ```
  a<b    →  zerorewrite(V_a-(V_b))<0
  !a     →  ! zerorewrite(V_a)
  ```

- 制御構造

  if 文、for 文、while 文、do-while 文といった制御構造は、同じ仕様であるため変更する必要はない。

- 型定義 typedef

  Asir 言語には、型の概念がないので必要ないが、新しい型名での型宣言を処理するための処理を行っている。

現在、未対応のものとその書換え方針は次のとおりである。

- 外部宣言

  局所変数と大域変数の有効範囲が C 言語とは異なるため、extern 宣言の追加や変数名の変更などの処理が必要である。

- 標準関数

  printf() など、C 言語の標準関数は個々に準備する必要がある。数値計算プログラムを対象とする場合、入出力の関数および数学関数が最低限必要である。

- 列挙定数

  大域変数にし、対応する整数を代入する。

- ビット演算子

  Asir に組込み関数として `iand()`, `ior()`, `ixor()`, `ishift()` などがあるためそれらへ変換する。しかし、整数型のデータ構造の違いに配慮が必要である。

- コンマ式

  Asir では、`for`、`while` の括弧中以外では、使用できないため書換えが必要である。

- 構造体

  Asir の構造体へ書換える。

- ポインタ

  配列参照のみの場合は、`*` を除くことで対応しているが、ポインタ参照が使用されている場合、対応が困難である。

- 配列の初期化

  `double d[] = {1.0, 2.0, 3.0};` などは、配列（ベクトル）の定義時に Asir の `newvect` 関数の引数に合致するように書換える必要があるため、これらは作成した `array` 関数で対応する。

- switch 文

  if-else 文に書換える必要がある。

- プリプロセッサ（CPP）

  現在は、プリプロセッサの処理後を入力としているが、処理前の`#define` などで書かれているマクロ等をそのまま処理できるようにする。

- 共用体と goto 文

  これらは対応できない。

- **(STAB4)** への対応

  再計算を容易に行うための補助プログラムを出力する。

3 例

本稿において実装したトランスレータの入力となる C 言語プログラムは、ポインタ参照を使っていないものに限定している。この制約を満たす標準的な数値計算プログラムとして文献 [5] のガウス・ジョルダン法のプログラムを取り上げる。この例の変換後のプログラムは、数値的に安定な入力に対し、正常に動作することを確認した。

3.1 言語の違いよる書換え

ここでは、C 言語と Asir 言語の違いを吸収するために書換えた部分を示す。次の入力の一部

```
void gaussj(float **a, int n, float **b, int m)
/* Linear equation solution by Gauss-Jordan elimination, ... */
{
    int *indxc,*indxr,*ipiv;
    int i,icol,irow,j,k,l,ll;
    float big,dum,pivinv,temp;

    indxc=ivector(1,n); /* The integer arrays ipiv, indxr, and indxc */
    indxr=ivector(1,n); /* are used for bookkeeping on the pivoting */

...

    free_ivector(indxr,1,n);
    free_ivector(indxc,1,n);
}
```

に対して、

```
def f_gaussj( V_a,  V_n, V_b,  V_m)
/* Linear equation solution by Gauss-Jordan elimination, ... */
{

    V_indxc=f_ivector(1,V_n); /* The integer arrays ipiv, indxr, and indxc */
    V_indxr=f_ivector(1,V_n); /* are used for bookkeeping on the pivoting */

...

    f_free_ivector(V_indxr,1,V_n);
    f_free_ivector(V_indxc,1,V_n);
}
```

と出力する。この結果では、関数定義は Asir 言語での def 文を用いるように書換えている。関数名には f_ が、変数名には V_ が加えられ、Asir では必要のない変数宣言は削除されているのが分かる。また、コメントはそのまま出力する。ここで、ivector、free_ivector は、文献 [5] 内で共通に使われているメモリ管理の関数である。

3.2 安定化手法のための書換え

ここでは、自動ゼロ書換え機能が有効である場合に、安定化手法のために書換えた部分を示す。次の入力の一部

```
                        if (fabs(a[j][k]) >= big) {
                            big=fabs(a[j][k]);
                            irow=j;
                            icol=k;
                        }
    ...
        if (a[icol][icol] == 0.0) nrerror("gaussj: Singular Matrix-2");
        pivinv=1.0/a[icol][icol];
        a[icol][icol]=1.0;
```

に対して、

```
    ...
                        if (f_fabs(V_a[V_j][V_k])-( V_big) >=0) {
                            V_big=f_fabs(V_a[V_j][V_k]);
                            V_irow=V_j;
                            V_icol=V_k;
                        }
    ...

        if (V_a[V_icol][V_icol] ==0) f_nrerror("gaussj: Singular Matrix-2");
        V_pivinv=tointval(10*10^(-1))/V_a[V_icol][V_icol];
        V_a[V_icol][V_icol]=tointval(10*10^(-1));
```

と出力する。最初の if 文では、0 との比較に書換え実現している。一方、2 番目の if 文では、元のプログラムが 0 との比較であったためにこのような書換えの必要がない。また、浮動小数点定数は有理数化し、tointval 関数を用いて区間数化する。このように、データを区間数にすると区間演算機能を有効にした Risa/Asir では以降の計算を自動的に区間演算で行う。

4 まとめ

C 言語プログラムの自動安定化システムの開発を行っており、簡単なガウス・ジョルダン法のプログラムでは、安定化手法を適用したプログラムに変換できた。また、変換されたプログラムは、数値的に安定な入力では正常に動作することを確認した。今後、文献 [5] にある標準的な数値計算プログラムについては変換できるようにシステムの開発を行っていく予定である。また、今後の課題として、出力されたプログラムがあらゆる入力に対して安定化されているかどうかの検討、数値的な安定性と安定化手法の関係、安定化手法の適用範囲などがある。

参 考 文 献

[1] Y. Kondoh, M.-T. Noda, A software system for algorithm stabilization technique, *Proceedings of the 7th Asian Technology Conference in Mathmatics*, ATCM Inc., pp.360–367, 2002.

[2] K. Shiraishi, H. Kai, M.-T. Noda, Symbolic-numeric computation of Wu's method using stabilizing algorithm, *Proceedings of ATCM2001*, ATCM Inc., USA, pp.444–451, 2001.

[3] H. Sekigawa and K. Shirayanagi, Automatic Algorithm Stabilization System, Josai Mathematical Monographs 2, NLA'99 Computer Algebra, pp. 159–168, 2000.

[4] K. Shirayanagi, M. Sweedler, A Theory of Stabilizing Algebraic Algorithms, *Tech.Rep.95-28*, Cornell Univ., pp.1–92, 1995.

[5] W.H.Press, B.P.Flannery, S.A.Teukolsky, W.T.Vetterling, *Numerical Recipes in C*, Cambridge University Press, 1988.